Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Автоматизированные системы обработки информации и управления»

Отчет

Лабораторная работа № 3

По курсу «Технологии машинного обучения»

ИСПОЛНИТЕЛЬ:

Горбатенко И.А.

Группа ИУ5-64

_____

"__"_____2020 г.

ПРЕПОДАВАТЕЛЬ:

Гапанюк Ю.Е.

_____

"__"_____2020 г.

Москва 2020

Лабораторная работа №3 по курсу "Технологии машинного обучения"

Горбатнко И.А. ИУ5-64

Цель лабораторной работы: изучение способов предварительной обработки данных для дальнейшего формирования моделей.

Задание:

Выбрать набор данных (датасет), содержащий категориальные признаки и пропуски в данных. Для выполнения следующих пунктов можно использовать несколько различных наборов данных (один для обработки пропусков, другой для категориальных признаков и т.д.) Для выбранного датасета (датасетов) на основе материалов лекции решить следующие задачи: обработку пропусков в данных; кодирование категориальных признаков; масштабирование данных.

Выполнение:

In [38]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
```

В качестве датасета будем использовать базу данных автомобилей с некоторыми пропущенными значениями.

In [39]:
```
data = pd.read_csv('cars.csv', sep=",")
```

In [40]:
```
data.shape
```

Out[40]:
```
(205, 26)
```

In [41]:
```
data.dtypes
```

Out[41]:
```
symboling              int64
normalized-losses    float64
make                  object
fuel-type             object
aspiration            object
num-of-doors          object
body-style            object
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                 float64
stroke               float64
compression-ratio    float64
horsepower           float64
peak-rpm             float64
city-mpg               int64
highway-mpg            int64
price                float64
dtype: object
```

Просмотрим датасет на наличие пропущенных значений:

In [42]:
```
data.isnull().sum()
```

Out[42]:
```
symboling             0
normalized-losses    41
make                  0
fuel-type             0
aspiration            0
num-of-doors          2
body-style            0
drive-wheels          0
engine-location       0
wheel-base            0
length                0
width                 0
height                0
curb-weight           0
engine-type           0
num-of-cylinders      0
engine-size           0
fuel-system           0
bore                  4
stroke                4
compression-ratio     0
horsepower            2
peak-rpm              2
city-mpg              0
highway-mpg           0
price                 4
dtype: int64
```

Проверим правильность загрузки данных:

In [43]:
```
data.head()
```

Out[43]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	..
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	..
1	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	..
2	1	NaN	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	..
3	2	164.0	audi	gas	std	four	sedan	fwd	front	99.8	..
4	2	164.0	audi	gas	std	four	sedan	4wd	front	99.4	..

5 rows × 26 columns

Обработка пропусков

1) Удаление колонок, содержащих пустые значения

In [44]:
```
data_new_1 = data.dropna(axis=1, how='any')
(data.shape, data_new_1.shape)
```

Out[44]:
```
((205, 26), (205, 19))
```

2) Удаление строк, содержащих пустые значения

In [45]:
```
data_new_2 = data.dropna(axis=0, how='any')
(data.shape, data_new_2.shape)
```

Out[45]:
```
((205, 26), (159, 26))
```

3) Заполнение всех пустых значений нулями

In [46]:
```
data_new_3 = data.fillna(0)
data_new_3.head()
```

Out[46]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	..
0	3	0.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	..
1	3	0.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	..
2	1	0.0	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	..
3	2	164.0	audi	gas	std	four	sedan	fwd	front	99.8	..
4	2	164.0	audi	gas	std	four	sedan	4wd	front	99.4	..

5 rows × 26 columns

"Внедрение значений" - импьютация (imputation)

Обработка пропусков в числовых данных

Выведем информацию по числовым колонкам, содержащим пустые значения

In [47]:
```
num_cols = []
total_count = 205
for col in data.columns:
    # Количество пустых значений
    temp_null_count = data[data[col].isnull()].shape[0]
    dt = str(data[col].dtype)
    if temp_null_count>0 and (dt=='float64' or dt=='int64'):
        num_cols.append(col)
        temp_perc = round((temp_null_count / total_count) * 100.0, 2)
        print('Колонка {}. Тип данных {}. Количество пустых значений {}, {}%.'.fo
```

```
Колонка normalized-losses. Тип данных float64. Количество пустых значений 41, 2
0.0%.
Колонка bore. Тип данных float64. Количество пустых значений 4, 1.95%.
Колонка stroke. Тип данных float64. Количество пустых значений 4, 1.95%.
Колонка horsepower. Тип данных float64. Количество пустых значений 2, 0.98%.
Колонка peak-rpm. Тип данных float64. Количество пустых значений 2, 0.98%.
Колонка price. Тип данных float64. Количество пустых значений 4, 1.95%.
```

In [48]:
```
data_num = data[num_cols]
data_num
```

Out[48]:

	normalized-losses	bore	stroke	horsepower	peak-rpm	price
0	NaN	3.47	2.68	111.0	5000.0	13495.0
1	NaN	3.47	2.68	111.0	5000.0	16500.0
2	NaN	2.68	3.47	154.0	5000.0	16500.0
3	164.0	3.19	3.40	102.0	5500.0	13950.0
4	164.0	3.19	3.40	115.0	5500.0	17450.0
...	...	...	...	...	...	...
200	95.0	3.78	3.15	114.0	5400.0	16845.0
201	95.0	3.78	3.15	160.0	5300.0	19045.0
202	95.0	3.58	2.87	134.0	5500.0	21485.0
203	95.0	3.01	3.40	106.0	4800.0	22470.0
204	95.0	3.78	3.15	114.0	5400.0	22625.0

205 rows × 6 columns

Запоминаем индексы строк с пустыми значениями:

In [49]:
```
flt_index = data[data['price'].isnull()].index
flt_index
```

Out[49]:
```
Int64Index([9, 44, 45, 129], dtype='int64')
```

Проверяем:

In [50]:
```
data[data.index.isin(flt_index)]
```

Out[50]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base
9	0	NaN	audi	gas	turbo	two	hatchback	4wd	front	99.5
44	1	NaN	isuzu	gas	std	two	sedan	fwd	front	94.5
45	0	NaN	isuzu	gas	std	four	sedan	fwd	front	94.5
129	1	NaN	porsche	gas	std	two	hatchback	rwd	front	98.4

4 rows × 26 columns

используем встроенные средства импьютации библиотеки scikit-learn:

In [51]:
```
data_num_price = data_num[['price']]
data_num_price.head()
```

Out[51]:

	price
0	13495.0
1	16500.0
2	16500.0
3	13950.0
4	17450.0

In [52]:
```
from sklearn.impute import SimpleImputer
from sklearn.impute import MissingIndicator
```

Проверяем:

In [53]:
```
indicator = MissingIndicator()
mask_missing_values_only = indicator.fit_transform(data_num_price)
mask_missing_values_only
```

Out[53]:
```
array([[False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [ True],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [ True],
       [ True],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
```

```
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
```

```
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[ True],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
```

```
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False]])
```

Используем различные методы:

In [54]:
```
strategies=['mean', 'median','most_frequent']
```

In [55]:
```
def test_num_impute(strategy_param):
    imp_num = SimpleImputer(strategy=strategy_param)
    data_num_imp = imp_num.fit_transform(data_num_price)
    return data_num_imp[mask_missing_values_only]
```

In [56]:
```
strategies[0], test_num_impute(strategies[0])
```

Out[56]:
```
('mean',
 array([13207.12935323, 13207.12935323, 13207.12935323, 13207.12935323]))
```

In [57]:
```
strategies[1], test_num_impute(strategies[1])
```

Out[57]:
```
('median', array([10295., 10295., 10295., 10295.]))
```

In [58]:
```
strategies[2], test_num_impute(strategies[2])
```

Out[58]:
```
('most_frequent', array([5572., 5572., 5572., 5572.]))
```

In [59]:
```
def test_num_impute_col(dataset, column, strategy_param):
    temp_data = dataset[[column]]

    indicator = MissingIndicator()
    mask_missing_values_only = indicator.fit_transform(temp_data)

    imp_num = SimpleImputer(strategy=strategy_param)
    data_num_imp = imp_num.fit_transform(temp_data)

    filled_data = data_num_imp[mask_missing_values_only]

    return column, strategy_param, filled_data.size, filled_data[0], filled
```

In [60]:
```
data[['price']].describe()
```

Out[60]:

	price
count	201.000000
mean	13207.129353
std	7947.066342
min	5118.000000
25%	7775.000000
50%	10295.000000
75%	16500.000000
max	45400.000000

In [61]:
```
test_num_impute_col(data, 'price', strategies[0])
```

Out[61]:
```
('price', 'mean', 4, 13207.129353233831, 13207.129353233831)
```

In [62]:
```
test_num_impute_col(data, 'price', strategies[1])
```

Out[62]:
```
('price', 'median', 4, 10295.0, 10295.0)
```

In [63]:
```
test_num_impute_col(data, 'price', strategies[2])
```

Out[63]:
```
('price', 'most_frequent', 4, 5572.0, 5572.0)
```

Обработка пропусков в категориальных данных

In [64]:
```
cat_cols = []
for col in data.columns:
    # Количество пустых значений
    temp_null_count = data[data[col].isnull()].shape[0]
    dt = str(data[col].dtype)
    if temp_null_count>0 and (dt=='object'):
        cat_cols.append(col)
        temp_perc = round((temp_null_count / total_count) * 100.0, 2)
        print('Колонка {}. Тип данных {}. Количество пустых значений {}, {}%.'.fo
```

Колонка num-of-doors. Тип данных object. Количество пустых значений 2, 0.98%.

In [65]:
```
cat_temp_data = data[['num-of-doors']]
cat_temp_data.head()
```

Out[65]:

	num-of-doors
0	two
1	two
2	two
3	four
4	four

In [66]:
```
cat_temp_data['num-of-doors'].unique()
```

Out[66]:
```
array(['two', 'four', nan], dtype=object)
```

In [67]:
```
cat_temp_data[cat_temp_data['num-of-doors'].isnull()].shape
```

Out[67]:
```
(2, 1)
```

Импьютация наиболее частыми значениями:

In [68]:
```
imp2 = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
data_imp2 = imp2.fit_transform(cat_temp_data)
data_imp2
```

Out[68]:
```
array([['two'],
       ['two'],
       ['two'],
       ['four'],
       ['four'],
       ['two'],
       ['four'],
       ['four'],
       ['four'],
       ['two'],
       ['two'],
       ['four'],
       ['two'],
       ['four'],
       ['four'],
       ['four'],
       ['two'],
       ['four'],
       ['two'],
```

In [69]:
```
np.unique(data_imp2)
```

Out[69]:
```
array(['four', 'two'], dtype=object)
```

Импьютация константой:

In [70]:
```
imp3 = SimpleImputer(missing_values=np.nan, strategy='constant', fill_value
data_imp3 = imp3.fit_transform(cat_temp_data)
data_imp3
```

```
       ['four'],
       ['four'],
       ['four'],
       ['four'],
       ['two'],
       ['two'],
       ['two'],
       ['four'],
       ['four'],
       ['four'],
       ['four'],
       ['four'],
       ['four'],
       ['four'],
       ['four'],
       ['four'],
       ['four'],
       ['four'],
       ['four'],
       ['four'],
       ['four'],
       ['two'],
       ['two'],
```

In [71]:
```
np.unique(data_imp3)
```

Out[71]:
```
array(['const', 'four', 'two'], dtype=object)
```

Преобразование категориальных признаков в числовые

In [72]:
```
cat_enc = pd.DataFrame({'c1':data_imp2.T[0]})
cat_enc
```

Out[72]:

	c1
0	two
1	two
2	two
3	four
4	four
...	...
200	four
201	four
202	four
203	four
204	four

205 rows × 1 columns

Кодирование категорий целочисленными значениями

In [73]:
```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

In [74]:
```
le = LabelEncoder()
cat_enc_le = le.fit_transform(cat_enc['c1'])
```

In [75]:
```
cat_enc['c1'].unique()
```

Out[75]:
```
array(['two', 'four'], dtype=object)
```

In [76]:
```
np.unique(cat_enc_le)
```

Out[76]:
```
array([0, 1])
```

In [77]:
```
le.inverse_transform([0, 1])
```

Out[77]:
```
array(['four', 'two'], dtype=object)
```

Кодирование категорий наборами бинарных значений

In [78]:
```
ohe = OneHotEncoder()
cat_enc_ohe = ohe.fit_transform(cat_enc[['c1']])
```

In [79]:
```
cat_enc.shape
```

Out[79]:
```
(205, 1)
```

In [80]:
```
cat_enc_ohe.shape
```

Out[80]:
```
(205, 2)
```

In [81]:
```
cat_enc_ohe
```

Out[81]:
```
<205x2 sparse matrix of type '<class 'numpy.float64'>'
        with 205 stored elements in Compressed Sparse Row format>
```

In [82]:
```
cat_enc_ohe.todense()[0:10]
```

Out[82]:
```
matrix([[0., 1.],
        [0., 1.],
        [0., 1.],
        [1., 0.],
        [1., 0.],
        [0., 1.],
        [1., 0.],
        [1., 0.],
        [1., 0.],
        [0., 1.]])
```

In [83]:
```
cat_enc.head(10)
```

Out[83]:

	c1
0	two
1	two
2	two
3	four
4	four
5	two
6	four
7	four
8	four
9	two

Масштабирование данных

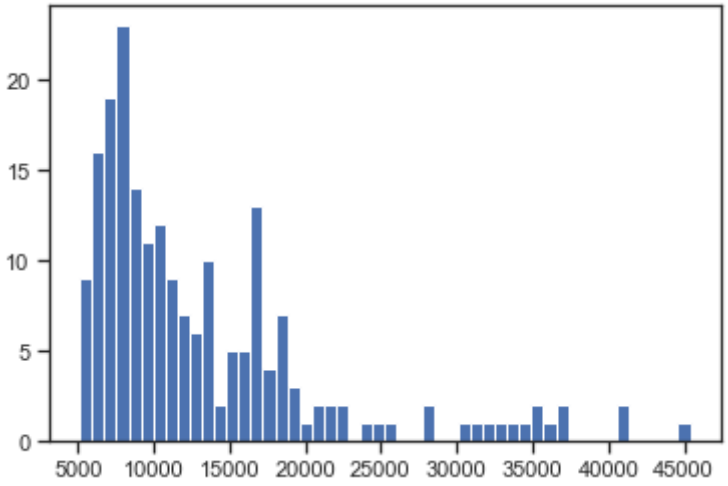
In [84]:
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler, Normalizer
```

MinMax масштабирование

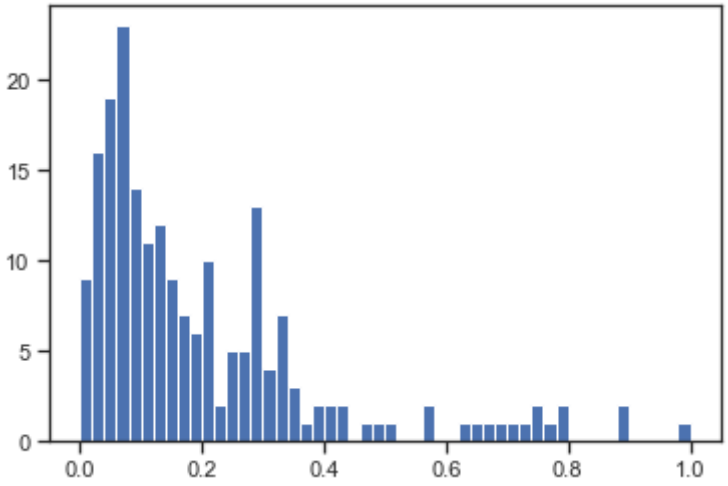
In [85]:
```
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data[['price']])
```

In [86]:
```
plt.hist(data['price'], 50)
plt.show()
```

```
/Users/Daria/opt/anaconda3/lib/python3.7/site-packages/numpy/lib/histogra
ms.py:839: RuntimeWarning: invalid value encountered in greater_equal
  keep = (tmp_a >= first_edge)
/Users/Daria/opt/anaconda3/lib/python3.7/site-packages/numpy/lib/histogra
ms.py:840: RuntimeWarning: invalid value encountered in less_equal
  keep &= (tmp_a <= last_edge)
```



In [87]:
```
plt.hist(sc1_data, 50)
plt.show()
```

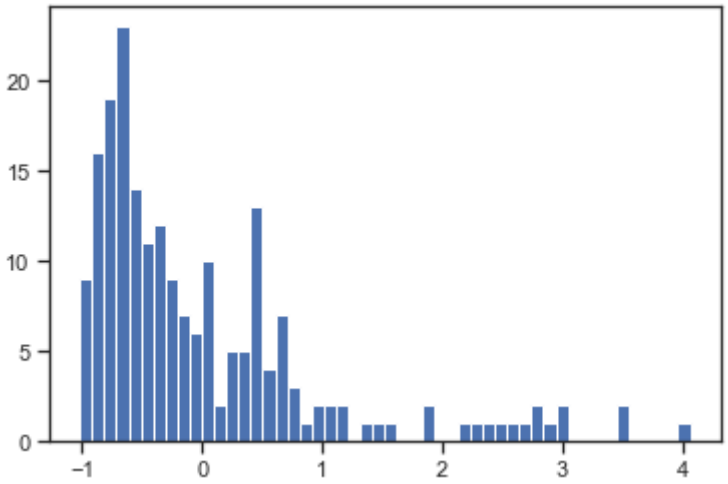


Масштабирование данных на основе Z-оценки

In [88]:
```
sc2 = StandardScaler()
sc2_data = sc2.fit_transform(data[['price']])
```

In [89]:
```
plt.hist(sc2_data, 50)
plt.show()
```

```
/Users/Daria/opt/anaconda3/lib/python3.7/site-packages/numpy/lib/histogra
ms.py:839: RuntimeWarning: invalid value encountered in greater_equal
  keep = (tmp_a >= first_edge)
/Users/Daria/opt/anaconda3/lib/python3.7/site-packages/numpy/lib/histogra
ms.py:840: RuntimeWarning: invalid value encountered in less_equal
  keep &= (tmp_a <= last_edge)
```



In [ ]:
```
```